Mining the Relationship between Bugs and Software Repositories

Boyuan Chen
Dept. of Computer Science and Engineering
York University
Toronto, Canada
chenfsd@yorku.ca

Ruoyu Gao
Dept. of Computer Science and Engineering
York University
Toronto, Canada
rgao@cse.yorku.ca

***Abstract*—In this paper, we conducted an experiment to predict file-level bugs using processed data metrics from software repositories. We compared three classification methods and four set of attributes on four Java-based projects. We found that balanced data can improve accuracy as well as recall of buggy file. Also, we found that using default configuration of logistic regression, recall of buggy file was 47.3% better in Weka than that in RapidMiner, 19.29% better than R. The model of cross project prediction performed poor than the model not crossed. However, attribute selection and resampling were not able to make the cross project model perform better.**

***Index Terms*— Bug, Data Mining, classification.**

I. Introduction

In modern software development procedure, there are mainly four steps: coding, testing, release and bug-fixing. In the first step, software developers construct functions of the project. Then these pieces of code are reviewed and handed to testing specialists. After testing phase, the new version is released. Then there are bug reports reported by users. After receiving the reports, project managers assign those bugs to specialists to do bug fixing.

For a new version, every project manager wants to make it less buggy as possible beforehand. And if one knows whether a file is more likely to be buggy or not, one can pay more or less attention to that file. Thus, we can save lots of effort for testing phase and make the software more robust.

This procedure is called bug prediction. There are already lots of studies focusing on this field. In a word, bug prediction is to predict a file is buggy or not by using a set of attributes of this file. For example, it could be lines of code (LOC), number of methods, etc. Furthermore, there are also some classification methods which could predict the severe level of those bugs, like critical bugs or trivial bugs.

In this paper, we tried to solve these four research questions.

RQ1: Given different sets of attributes, how will the performance vary between different classification methods?

RQ2: How will the performance vary if we resample the training data (since most data in bug prediction is skewed)?

RQ3: How will the performance vary when we do cross project validation?

RQ4: How will the performance vary if we used three different implementations of these methods?

RQ5: Can we achieve better results of cross project prediction by attribute selection?

For the first question, in bug prediction, there are many attributes which can be used. In this paper, we used four different sets of attributes, which are bug metrics, CK-OO metrics, churn of CK-OO metrics and entropy of CK-OO metrics.

Bug metrics is sets of attribute which indicates previous defects. CK represents Chi- damber & Kemerer suite [2] and OO represents object oriented metrics. Many bug prediction approaches are based on these metrics. The churn of metrics is used by Nikira et al. [4]. Hassan predicted defects using the entropy (or complexity) of code changes [3].

We compared three different classification methods as well, which are random forest [5], logistic regression [6] and decision tree. All of them are mature classification methods.

For the second question, most files do not tend to be buggy based on what we found. This leads to skewed data. Skewed dataset may lead to high accuracy but often low recall for the minority class (e.g. buggy class in this study). So we used the resampling method in Weka to preprocess the data and do the exact same experiment as we do for RQ1 to see the effect of balancing training data.

For the third question, usually each project needs to be predicted using the model trained on its own dataset. What if we use the same model to predict every other project? This test procedure is called cross-project validation. We will both do cross project validation with resampling and without it. Then we compare the performance with the results from RQ1 and RQ2.

There are many data mining tools that implemented all kinds of classification tools, for example, R, Weka and RapidMiner, etc. Although the theoretical basis of these classification methods are the same, the difference of implementation may impact the performance. We chose logistic regression to test.

The last question is about attribute selection. We selected one set of attribute to pick up some important attribute to do prediction to see the performance variance.

The rest of the paper are organized as follow: part two introduces the related work done by previous studies; part three introduces the approach we used in this study, describes the procedure in general and illustrates examples; part four

introduces experiment in detail and analyzes the results; part five gives the conclusion; part six gives the reference and part seven is appendixes.

II. Related work

Defect prediction played an important role in modern software development process. Therefore there are many bug prediction methods. D'Ambros et al. [1] provided a benchmark to evaluate defect prediction.

Bug prediction is usually file level, thus attributes in file level are needed.

One of the famous attribute set used is Chidamber and Kemerer (CK) metrics suite [2]. There is also an additional set of attributes called object-oriented metrics.

Hassan introduced the concept of entropy of changes, a measure of the complexity of code changes [3]. Entropy was compared to the amount of changes and the amount of previous bugs.

D'Ambros et al. [1] also introduced attribute sets of bug metrics, which contained bug categories of: all bugs, non trivial bugs (severity>trivial), major bugs (severity>major), critical bugs (critical or blocker severity) and high priority bugs (priority>default).

Nikora et al. [4] used churn of CK and OO metrics to predict post release defects. They sampled the history of source code every two weeks and calculated the deltas of source code.

III. Approaches

A. Description of Method

In this experiment, we mainly did four types of experiments for five research questions.

Firstly, we compared three classification methods using four sets of metrics on four projects. The four projects are eclipse, lucene, mylyn and pde which are all Java-based. The four sets of metrics are single version CK/OO metrics, bug metrics, churn metrics and entropy metrics. The details of calculation for these metrics were introduced in [1]. We put these four sets of dataset into Weka and do a ten-fold validation.

Secondly, having found that dataset was skewed, we resampled the raw data. We arbitrary made the ratio between not buggy and buggy 0.6. After resampling, we trained the model using three classification methods then used the origin data as test data.

Thirdly, we randomly picked lucene project to do cross-project validation. We used the same metrics (e.g bug metrics) in one project (lucene) to train a model and predict using the same metrics in other projects (eclipse, mylyn and pde) and all three kinds of classification methods. We also compared the performance between cross-project prediction and original ones using resampled data.

Fourthly, for Random Forest and Logistic Regression, there are two implementation packages in Weka, R and RapidMiner. We used results from the third experiment to compare with R. By that case, we used sample data in R and tried to see if there were any differences between two implementations.

At last, we chose CK-OO of 4 projects to do attribute selection and LOG as classification method in order to see if attribute selection could have a significant improvement.

B. Example illustration

First we preprocessed the data. In order to be general, we only made two classes, buggy or not buggy. When we put the data into Weka, it can be seen in *Figure 1*. We have 5 attributes: number of bugs found until, number of nontrivial bugs found until, number of major bugs found until, number of critical bugs found unitil, number of high priority bugs found until.

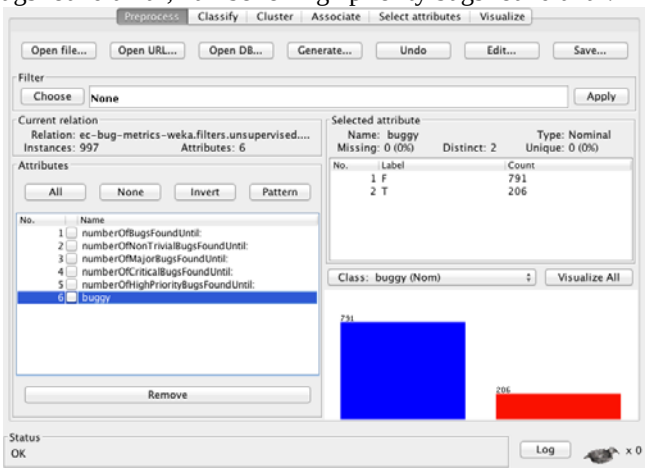

Figure 1. Put data in Weka

Then we applied three classification methods on it using ten-fold validation. *Figure 2* shows the results of random forest classification.

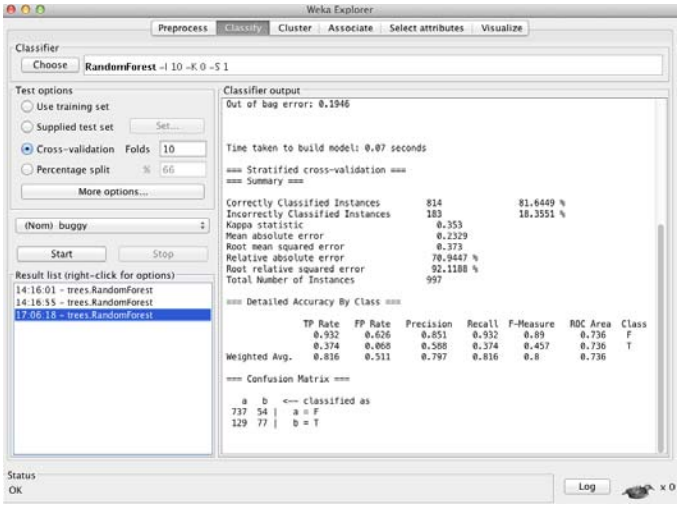

Figure 2. Running Result

We recorded the recall of class T, the classification accurcy in a worksheet file. Then we used statistical analysis to analyze the results.

IV. Experiment results

A. Description of the Experiments

For the experiments, we put the processed data into Weka, and for each project, we used three classification methods:

random forest, logistic regression and decision tree on four sets of metrics.

B. Results and Discussion

RQ1: Given different sets of attributes, how will the performance vary between different classification methods?

For the evaluation results of different classification methods, it can be seen in *Figure 3*.

As we can see, logistic regression had the highest accuracy (87.21%) and random forest had the highest recall of True value (22.54%).

The effect of classification methods is not statistical significant with $p > 0.05$. And the effect of sets of attributes is not statistical significant with $p > 0.05$ as well.

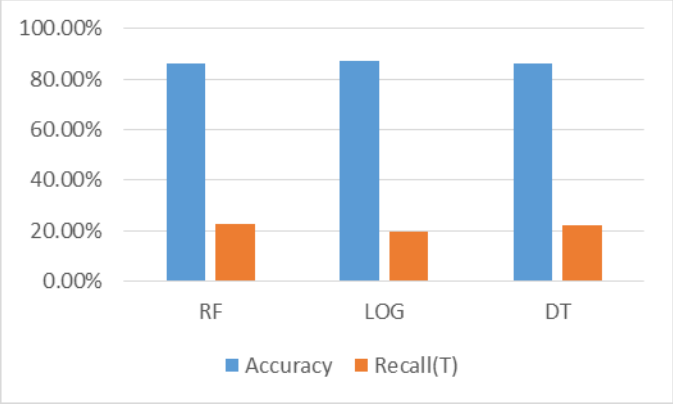

Figure 3. Accuracy and recall of each classification method

For the evaluation of different sets of attributes, the result can be seen in *Figure 4*.

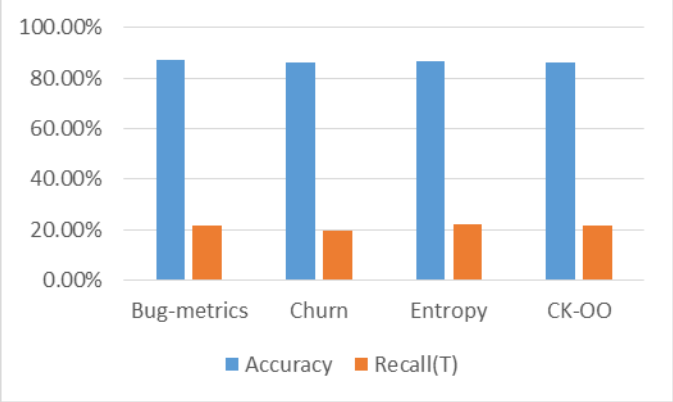

Figure 4. Accuracy and recall of each set of attributes

From the results we can see, although all the accuracy is high in the tables above, the recall is very low. In real life, the software development manager will pay more attention to the situation of predicting right the bugs. So we picked the recall of class buggy to evluate. This low recall maybe due to the skewness of the dataset. For example, in the dataset of lucene, the not buggy class has 627 instances while buggy has only 64 instances. So we consider resampling the data in order to gain high recall.

RQ2: How will the performance vary if we balance the training data (since most data in bug prediction is skewed)?

Firstly we tried different ratio of resampling the data. By using the resampled data as training data and the original data as testing data, we chose 0.6 because the accuracy dropped when the ratio went higher.

As it can be seen in *Figure 5*, the accuracy did not vary remarkably. The highest mean accuracy came from the RF method, which was 90.20%, and the accuracy of DT and LOG were 87.02% and 84.39% respectively. Not surprisingly, the effect of resampling on the classification accuracy of these classification methods was significant ($p < .005$). Using LSD method, we found that the accuracy of RF was significantly better than the accuracy of LOG and DT.

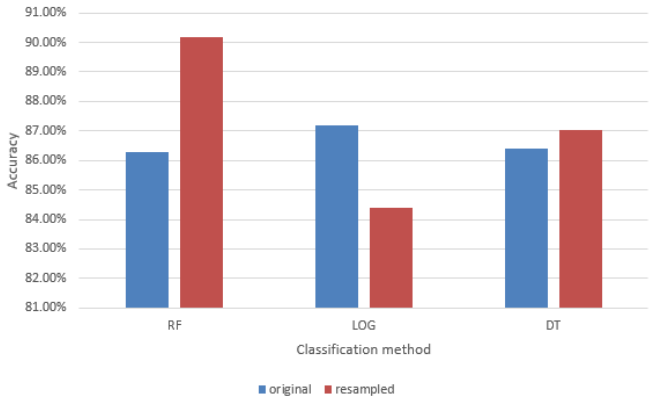

Figure 5. Accuracy of each classification method

The average recall of "buggy" of each method increased at least 142%. The recall of RF was the highest which was 0.82, and the lowest was 0.47 from LOG. Besides, the difference between recall of using original data and using resampled data as training data was significant ($p < 0.05$). See *Figure 6*.

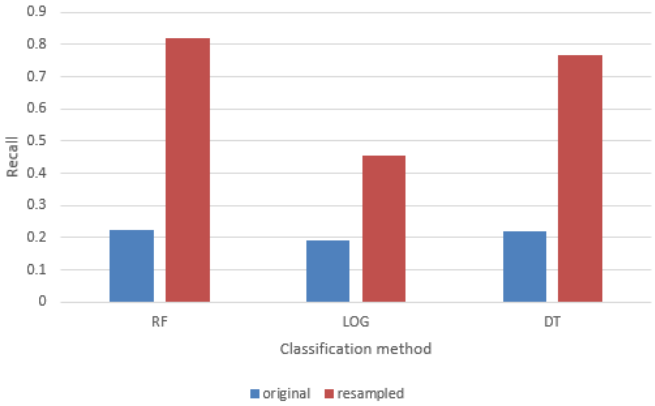

Figure 6. Recall of each classification method

For skewed dataset, classification methods could fail to classify the minority although the accuracy could be high, wheras the class with low occurrence sometimes was more important.

In bug prediction, actually the cost of misclasification was hard to define. For example, if a class was buggy but was predicted as not buggy, the influence of the bug(s) in the file would only be revealed after the software was released. While if a file contained no bug but was predicted as buggy,

developers might take more effort because they would try to find bugs in a class which did not have bug. As a result, developers would review or test the file more times to varify if the bugs were covert.

So in the following sections, we considerd accuarcy as the first evaluation critiron. If the accuarcy among different methods or different metrics was not significant, we would compare the recall of "buggy" (denoted as True) because bugs would have a higher chance to be fixed as long as the file which contained bugs was classified as "buggy", and files labeled with "buggy" were relatively hard to be classified.

In terms of set of attributes, the mean accuracy of CK-OO was 89.72% which was the highest and the lowest mean accuracy came from bug-metrics (83.80%). See Figure 7. The difference of these set of attributes were statistically significant ($p < .05$).

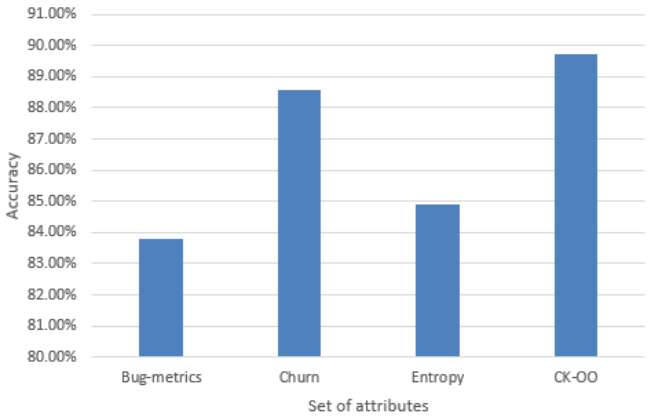

Figure 7. Accuracy using each set of attributes

Using LSD method, we found that the accuracy of ck-oo method was significantly better than those of bug-metrics and entropy. Though the accuracy of Churn was significant better than that of Bug-metrics, the difference among it and Entropy was not significant.

Besides, the difference of recall among different methods was not significant ($p > .05$).

From these, we concluded that using metrics directly related to code such as lines of code or changes of lines of code could achieve the best result, whereas predicting bugs based on previous defects (Bug-metrics) was not reliable: bugs didn't have the Matthew effect.

RQ3: How will the performance vary when we do cross-project validation?

First part: We then did cross-project validation. We arbitrary trained the model from lucene project without resampling.

In this analysis, we ignore effect of set of attributes. The results can be seen in *Figure 8*.

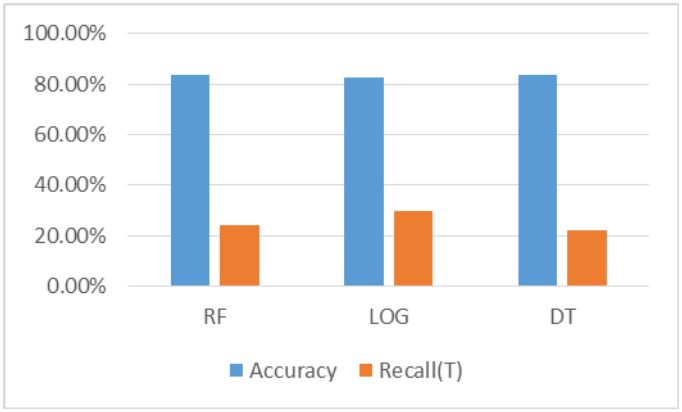

Figure 8. Accuracy and recall of each classification method

The mean accuracy of three methods are 83.81%, 82.56% and 83.57%. And the effects of cross-project validation on all of three methods are not statistically siginificant. This may be also due to the skewness of the dataset.

It could be seen that for random forest and decision tree classification methods, the effect of cross-project validation is not statistical significant with t-value > 0.05. The mean recall for these two methds are 24.22% and 22.26%. For logistical regression, the effect of cross-project validation is statistical significant with t-value = 0.004. The recall of class buggy is 29.97%, in comparison with 19.36% before.

Then we resampled the data, using logistic regression as the classification methods to do the cross-project validation. The results can be seen in the Figure 9.

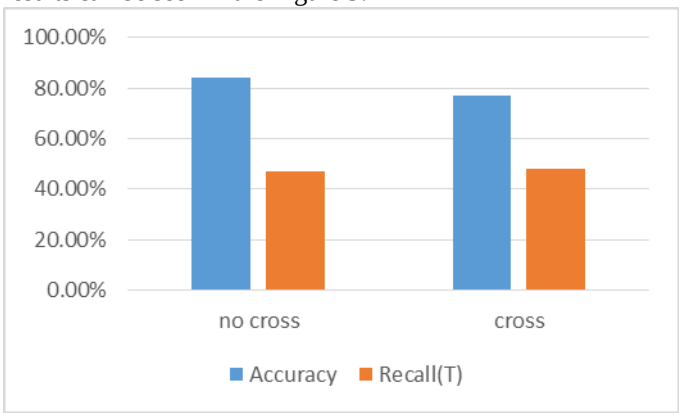

Figure 9. Accuracy and recall using cross-project and non-cross-project

The original mean accuracy for logistic regression is 84.39%, while cross-project mean accuarcy is 77.27%. The effect of cross-project is significant with t-value =0.001. This means that the accuracy is lower by doing cross-project prediction.

And for recall, the origin mean value is 46.98% and the cross-project mean value is 48.06%. However, the effect of cross-project is not statistical significant with t-value > 0.05.

This result means that cross-project prediction is not as good as the origin ones since the accuracy dropped and no improvements on recall.

RQ4: How will the performance vary if we used three different implementation of these methods?

We picked the LOG method as classification method, the resampled data set of eclipse as training data set and the original data set of eclipse as testing data set for the purpose of evaluation.

Because the options provided in the configuration of LOG in these three tools were different (for example RapidMiner can set kernel type but Weka can't), we used the default configuration to evaluate these tools.

As it shown in Figure 10, the accuracy varied slightly among three tools, and R had the highest mean accuracy. But the difference was not significant ($p > .05$). Although it was not significant different, the *p* value was quite close to the threshold which was 0.053.

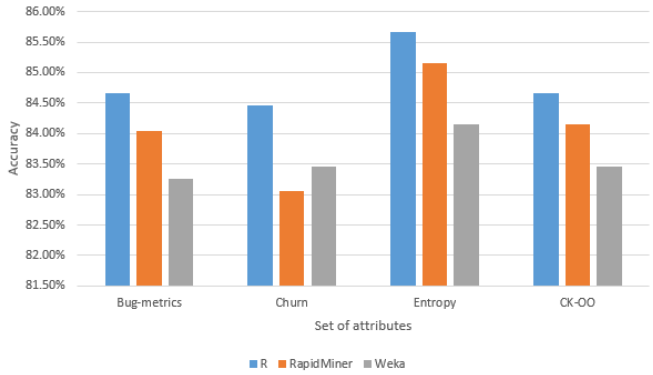

Figure 10. Accuracy using each set of attributes

In contrast, the difference of recall was significant ($p < 0.01$). As can be seen in Figure 11, LOG in Weka had the highest recall for each set of attributes, and LOG in RapidMiner had the lowest recall for each set of attributes. The mean recall for LOG in Weka was 0.61, and the mean recall for R and RapidMiner was 0.51 and 0.41 respectively.

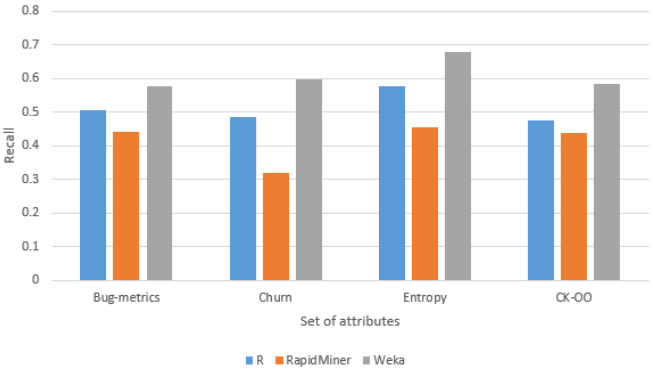

Figure 11. Recall using each set of attributes

The classification results could be different because of different implementations and different configurations. Weka provided the least flexibility of setting parameters of LOG, but it had the highest accuracy and recall. R as a script language offered much more options, and could reach considerable accuracy with default options. But in terms of user-friendly, R was not as good as Weka and RapidMiner. Besides, because the dataset was relatively small, the execution speed of all three tools were fast.

RQ5: Can we achieve better results of cross project prediction by attribute selection?

Based on the previous sections, we chose CK-OO of 4 projects to do attribute selection and LOG as classification method in order to see if attribute selection could have a significant improvement.

We ranked the attributes in CK-OO by Chi-square method in Weka for each project, and selected 8 attributes based on the average rank of the attributes. See Table 1.

Table 1. Rank of each attribute

	eclipse	lucene	mylyn	pde	average rank
rfc	3	6	3	1	3.25
fanOut	4	5	1	3	3.25
cbo	5	1	4	6	4
numberOfLinesOfCode	2	8	5	2	4.25
wmc	1	7	6	4	4.5
numberOfMethods	6	2	10	8	6.5
lcom	7	3	9	9	7
numberOfPrivateMethods	9	15	2	5	7.75

We selected the CK-OO from Lucene as training dataset because this was the largest dataset. However, both the differences of accuracy and recall between before the selection and after the selection were not significant ($p > .05$).

V. Conlusions

In this paper, we mined the relationship between bugs and bug repositories. We did bug prediction on four java-based projects using four sets of attributes and three classification methods.

We mainly answered five research questions.

First of all, we found that the effects of sets of attributes and classification methods were not statistically significant on accuracy of prediction and the recall of buggy class.

Then we balanced the dataset, ran the same experiment again, the effect of resampling is significant in accuracy and recall. Both accuracy and recall are higher than the original ones.

Then we ran cross project prediction on original data, there was no significant difference between them. After resampling, we ran the cross project prediction using logistic regression again, the accuracy dropped while recall remained no significant difference.

For different implementation of logistic regression in R, Weka and RapidMiner, the effect on accuracy was not significant different while Weka accomplished the highest value of recall.

At last, we did attribute selection of CK-OO metrics to do cross project prediction, both the differences of accuracy and recall between before the selection and after the selection were not significant.

References

[1] D'Ambros, Marco, Michele Lanza, and Romain Robbes. "Evaluating defect prediction approaches: a benchmark and an extensive comparison." Empirical Software Engineering 17.4-5 (2012): 531-577.

[2] Shyam R. Chidamber and Chris F. Kemerer. A metrics suite for object oriented design. IEEE Trans. Software Eng., 20(6):476–493, 1994.

[3] Ahmed E. Hassan. Predicting faults using the complexity of code changes. In Proceedings of ICSE 2009, pages 78–88, 2009.

[4] Allen P. Nikora and John C. Munson. Developing fault predictors for evolving software systems. In Proceedings of the 9th International Symposium on Software.

[5] Liaw, Andy, and Matthew Wiener. "Classification and Regression by randomForest." R news 2.3 (2002): 18-22.

[6] Hosmer, David W., Stanley Lemeshow, and Rodney X. Sturdivant. Introduction to the logistic regression model. John Wiley & Sons, Inc., 2000.